SAME: Skeleton-Agnostic Motion Embedding for Character Animation

SUNMIN LEE, Seoul National University, South Korea
TAEHO KANG, Seoul National University, South Korea
JUNGNAM PARK, Seoul National University, South Korea
JEHEE LEE, NC Research, Seoul National University, South Korea
JUNGDAM WON*, Seoul National University, South Korea

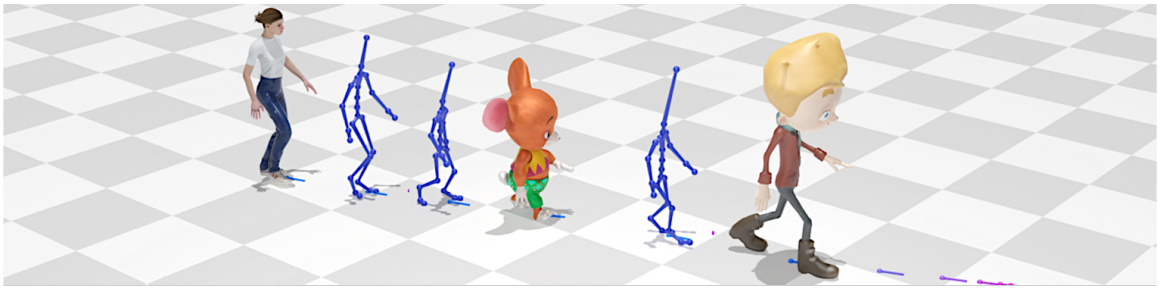

Fig. 1. Our skeleton-agnostic motion embedding allows us to develop *Motion Matching* controller using a collection of heterogeneous motion datasets. Additionally, our framework enables on-the-fly morphing to other characters by easily changing the target skeleton.

Learning deep neural networks on human motion data has become common in computer graphics research, but the heterogeneity of available datasets poses challenges for training large-scale networks. This paper presents a framework that allows us to solve various animation tasks in a skeleton-agnostic manner. The core of our framework is to learn an embedding space to disentangle skeleton-related information from input motion while preserving semantics, which we call Skeleton-Agnostic Motion Embedding (SAME). To efficiently learn the embedding space, we develop a novel autoencoder with graph convolution networks and provide new formulations of various animation tasks operating in the SAME space. We showcase various examples, including retargeting, reconstruction, and interactive character control, and conduct an ablation study to validate design choices made during development.

CCS Concepts: • **Computing methodologies** → **Animation**; **Motion processing**; **Motion capture**; **Neural networks**; **Learning latent representations**.

Additional Key Words and Phrases: Character Animations, Motion Retargeting, Motion Embedding, Graph Neural Networks

ACM Reference Format:
Sunmin Lee, Taeho Kang, Jungnam Park, Jehee Lee, and Jungdam Won. 2023. SAME: Skeleton-Agnostic Motion Embedding for Character Animation. In *SIGGRAPH Asia 2023 Conference Papers (SA Conference Papers '23), December 12–15, 2023, Sydney, NSW, Australia.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3610548.3618206

*corresponding author

SA Conference Papers '23, December 12–15, 2023, Sydney, NSW, Australia

1 INTRODUCTION

Learning deep neural networks on human motion data has become a common practice in computer graphics research. Since the early 90s, human mocap data have been collected from various sources and are often readily available online. However, using all available datasets to train a large-scale network is still challenging due to their heterogeneity, where the characters in the datasets often have different skeletal structures, body proportions, joint types, and joint coordinate systems. Unlike other modalities such as images, videos, or audio, converting between different motion representations is not straightforward but requires a non-trivial procedure called *retargeting*, which involves solving non-linear optimization problems.

The AMASS dataset [Mahmood et al. 2019], which recently became available to the graphics and vision communities, has converted many existing heterogeneous datasets into a unified data format based on SMPL [Loper et al. 2015]. Although this unified dataset has paved the way for building large-scale deep neural network models for understanding and generating human motion, there are several technical and practical limitations to the network model learned from this kind of unified datasets. First, it can only generate motions with the same skeleton, which limits the applicability of the learned model in various graphics applications. For example, a computer game may have many characters with different skeletons. Having multiple skeleton-specific network models for individual characters is inefficient and inconvenient. Second, the data format relies on SMPL, which was built based on human body scans. Therefore, it is cumbersome to use the network models based on SMPL for characters with significantly different body proportions from normal humans.

In this paper, we present a new framework that allows motion data to be collected, represented, edited, and employed in various animation tasks regardless of their skeletal structure (e.g. body proportions, limb lengths, and the number of joints). From the training

data collected from various sources, which may include motion data with diverse skeletons, we learn a **S**keleton-**A**gnostic **M**otion **E**mbedding (SAME in short) for biped characters by using a novel autoencoder structure based on graph convolution networks, which can efficiently disentangle skeleton-relevant information from the input motion while preserving its original semantics. Our framework performs animation tasks in the SAME space, where we propose new formulations for popular animation tasks such as motion classification, retargeting, missing joint reconstruction, motion similarity, and interactive character control. Furthermore, the ablation studies to verify the our design choices are also demonstrated. Code for the paper is available at https://github.com/sunny-Codes/SAME.

2 RELATED WORK

2.1 Skeletal Variation

Researchers have explored methods to address skeletal variations in character animation, with a fundamental procedure known as retargeting, which involves transforming motions from a source character to match a target character. It is formulated as non-linear trajectory optimization with spatial and temporal constraints, and efficient algorithms have been proposed [Choi and Ko 1999; Gleicher 1998; Lee and Shin 1999; Shin et al. 2001; Sturman 1998; Tak and Ko 2005]. Recently, several deep learning-based approaches for retargeting have been introduced, such as retargeting in the unsupervised setting [Aberman et al. 2020; Lim et al. 2019; Villegas et al. 2018]. Aberman et al. [2020] proposed a method for retargeting among different homeomorphic skeletons which may have the same set of end-effectors but different numbers of joints by constructing a shared latent space among homeomorphic skeletal motions. Our method differs in that we build a single network to address skeletal variety while their method requires a separate network to be trained for each skeletal structure. Aside from retargeting, skeletal variation has also been explored in other motion generation models. Hou et al. [2021] suggested motion synthesis for the skeleton parameterized by sizes and proportions, Li et al. [2021] developed a general-purpose human motion prior that accommodates varying skeletons. In physics-based character control, it is often utilized to produce physically plausible motions given skeletal structures and physical properties [Park et al. 2022; Won and Lee 2019].

2.2 Motion Latent Space

Manipulating motions directly is often challenging because they are high-dimensional time-series data. Constructing compact representations has gained attention for a long time not only in computer graphics but also in other domains such as robotics and computer vision. Prior to the deep learning era, attempts were made using Principal Component Analysis [Chai and Hodgins 2005; Safonova et al. 2004; Shin and Lee 2006] and Gaussian Process Latent Variable Models [Wang et al. 2007] to construct low-dimensional motion embeddings. Holden et al. [2015] first applied deep learning based on convolutional auto-encoder and demonstrated several applications like motion denoising, similarity search, and interpolation within the embedding space. Aberman et al. [2019] presented character-agnostic latent space specialized for 2D motion retargeting. Starke et al. [2022] proposed learning periodic embeddings to capture the spatio-temporal structure of motion data.

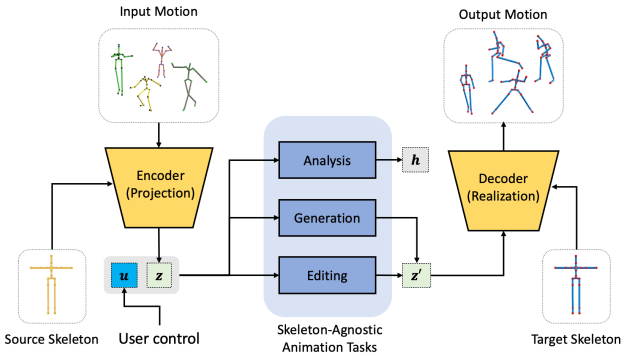

Fig. 2. System overview.

Recently, there have been continuous efforts to build a universal and generalizable motion embedding, often called motion prior [Ghorbani et al. 2020; Habibie et al. 2017; Ling et al. 2020], inspired by the success of large-scale deep neural networks in other domains like images and natural language processing [Brown et al. 2020; Chowdhery et al. 2022; Touvron et al. 2023]. The recently introduced AMASS dataset [Mahmood et al. 2019], which unified more than 40 hours of existing datasets into SMPL [Loper et al. 2015; Pavlakos et al. 2019] formats, has further sparked the research direction of learning motion prior [He et al. 2022; Raab et al. 2023; Rempe et al. 2021; Yuan et al. 2022]. However, they are applicable only to normal humans with a fixed skeleton topology, which poses limitations when applying the learned models to graphics applications where similar but different topologies may exist even in the same application. We present a framework that enables users to perform various animation tasks at an abstract level, with the flexibility to retarget the output motions to specific downstream applications as needed.

2.3 Graph Based Motion Processing

Our novel autoencoder relies on Graph Neural Networks (GNN), which extend the convolution operation to non-grid data structures represented by graphs. Yan et al. [2018] proposed a spatiotemporal graph convolution model for action recognition, which has been further extended in many other applications [Huang et al. 2020; Jang et al. 2022; Li et al. 2019; Shi et al. 2019]. Xu et al. [2020] developed a model that generates animation rigs from 3D character meshes using the edge convolution network suggested by Wang et al. [2019]. Aberman et al. [2020] suggested a graph-based skeleton operation to construct a latent space for different yet homeomorphic skeleton structures. They learned a separate kernel for each body joint, requiring a separate network for each skeletal topology. In contrast, our framework learns a shareable kernel for all joints, allowing a single network model to handle arbitrary skeletal topologies.

3 OVERVIEW

Consider a general character animation task T. Given a single pose, a motion clip, or a motion dataset as input, one can perform analysis

tasks (e.g. similarity measure, classification, clustering), editing tasks (e.g. retargeting, denoising), or generation tasks (e.g. interactive control, inbetweening). These tasks can be regarded as mappings that falls under either $T : (M, u) \rightarrow (h)$ or $T : (M, u) \rightarrow (M)$, where M, u, and h stand for motion, user control, and analysis features, respectively. For example, measuring the similarity between two motions is a mapping from two input motions to a scalar value, and interactive character control is a sequential mapping from the motion generated so far and the current user control (e.g. joystick input) to a new motion. The fundamental concept behind using deep learning to solve these tasks is to construct such mappings using deep neural networks. The underlying assumption that most existing methods rely on is that the skeletal structure remains the same. Consequently, once a mapping is constructed, it can only consume or generate motions with the same skeletal structure used in the training procedure. This conventional assumption restricts us from utilizing various datasets available online, as they are often recorded with different skeletal structures from each other. Our objective is to address this skeleton-dependency problem by solving general animation tasks in a skeleton-agnostic manner.

Figure 2 illustrates an overview of our framework. We first project (encode) an input motion, whose representation is tied with a specific skeleton, into the **S**keleton-**A**gnostic **M**otion **E**mbedding (SAME) space, where we perform general animation tasks. For instance, we can measure the similarity between two motions in the embedding space, even if their skeletal structures differ. In motion editing or generation tasks, we initially perform the task in the SAME space and then realize (decode) the processed embeddings as motions by conditioning target skeletons. During this step, our framework provides the functionality to decode with arbitrary skeletons, ensuring that the generated motions align with various target applications.

4 SKELETON-AGNOSTIC MOTION EMBEDDING

4.1 Data Representation

Skeletal motion data $M = (S, D^{1:T})$ generally consists of skeleton data S and motion data D. The skeleton data represents the skeletal structure of a character, which is defined by $S = \{p_{1:J}, o_{1:J}\}$ where J is the number of joints of the character, $p_j \in \mathbb{R}^3$ is the absolute joint position and $o_j \in \mathbb{R}^3$ is the relative joint offset with respect to its parent joint. It is worth noting that even for the same characters, S could be different if they use different coordinate systems. Therefore, we perform standardization beforehand where we express joint local coordinates with respect to its coordinate at T-pose. The process is as follows. We first make the character's pose T-pose and align its facing direction with the z-axis of the default coordinate system while adjusting its height so that its feet touch the ground plane (XZ plane). Then, we implant a new joint coordinate system $H'_j = (I, p_j)$ for each joint, where $I \in R^{3x3}$ is the identity matrix and $p_j \in R^3$ is the global joint position. The process is the same operation as "reset transformation" equipped in many 3D animation software.

The motion $D^{1:T}$ is a time series data, where T is the number of frames and is defined as $D^t = \{q^t_{1:J}, p^t_{1:J}, p^{t-1}_{1:J}, v^t_{1:J}, r^t, c^t_{1:J}\}$ at time frame t. We use the first two columns of rotation matrix [Zhou et al. 2019] to represent joint rotation $q_j \in \mathbb{R}^6$ with respect to its parent coordinate system. Joint positions are computed in the character's facing frame [Holden et al. 2017]. The movement of the root joint is computed by $r^t = (\Delta x, \Delta z, \Delta\theta, h)$, where $(\Delta x, \Delta z)$ and $\Delta\theta$ are translation and rotation on the ground represented in the facing frame at the previous frame, and h is the absolute height from the ground. Contact labels $c^t_{1:J}$ indicate whether joints are in contact with the ground or not. The labels are automatically computed using a simple heuristic based on joint height and velocity thresholds [Lee et al. 2002].

4.2 GNN Autoencoder

In most deep learning-based approaches, the skeleton data was often disregarded under the assumption that the skeletal structure remains unchanged. However, we take advantage of both skeleton and motion data to efficiently address various animation tasks in a skeleton-agnostic manner. This poses a challenge due to the variable dimensionality of the data. To overcome this, we propose a novel autoencoder based on graph convolution networks (GCN). In our architecture, character joints and their connections are represented as nodes and edges in an input graph. This enables the encoder to generate motion embeddings that are independent of the specific skeleton structure.

Before discussing our novel autoencoder in detail, we first explain the basic mechanism of Graph Convolution Networks (GCN) and the operations used in our model. GCN learns meaningful features from input graph data through message passing, where node features are exchanged through graph edges, and each node's feature is updated based on spatial rules. We adopt the method proposed by Veličkovi'c et al. [2017], which incorporates attention mechanisms for feature updates. Given a node (joint in our data) feature $\mathbf{x}_i$, the spatial rule to update a new feature $\mathbf{x}'_i$ is as follows:

$$\mathbf{x}'_i = \alpha_{i,i}\mathbf{\Theta}\mathbf{x}_i + \sum_{j \in \mathcal{N}(i)} \alpha_{i,j}\mathbf{\Theta}\mathbf{x}_j \tag{1}$$

$$\alpha_{i,j} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^\top[\mathbf{\Theta}\mathbf{x}_i \,\|\, \mathbf{\Theta}\mathbf{x}_j]\right)\right)}{\sum_{k \in \mathcal{N}(i)\cup\{i\}} \exp\left(\text{LeakyReLU}\left(\mathbf{a}^\top[\mathbf{\Theta}\mathbf{x}_i \,\|\, \mathbf{\Theta}\mathbf{x}_k]\right)\right)} \tag{2}$$

where $N(i)$ represents the set of neighbor indices, x_j is the feature of the j-th neighbor, and $\|$ denotes vector concatenation. $\mathbf{\Theta}$ and $\mathbf{a}$ are shared learnable parameters for all nodes, representing a linear feed-forward layer and feature scaling weights, respectively. The feature is updated by a linear combination of its own feature and the features of its neighbors, weighted by attention weights $\alpha_{i,j}$ that determine each neighbor's contribution. Prior to the combination, the features undergo transformation by $\mathbf{\Theta}$. To enhance expressive power, we incorporate multiple channels (multi-headed attention) where several sets of $\mathbf{\Theta}$ and $\mathbf{a}$ are learned concurrently. Additionally, *Graph Pooling* is often integrated into GCN, aggregating node features and generating a fixed-length vector. In our model, we use max pooling. Figure 5 depicts the convolution and pooling operations applied to an input skeletal pose.

Figure 3 depicts the structure of our novel autoencoder, designed to generate embeddings for input motions on a frame-by-frame basis, enabling the processing of time-varying inputs. The motion embedding $z^{1:T}$ is constructed by encoding each frame of the input motion. The encoder takes the source skeleton and pose data at each frame as input, producing $z^t = \text{Enc}(S_{\text{src}}, D^t_{\text{src}})$, which represents a

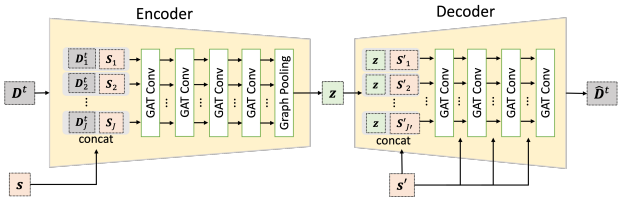

Fig. 3. Our autoencoder based on graph convolution networks to process motions in a skeleton-agnostic manner.

fixed-size, skeleton-agnostic pose embedding. This encoding process involves multiple graph attention convolution layers followed by graph max pooling. The decoder takes a target skeleton S_{tgt} and a skeleton-agnostic pose embedding z^t as input, generating pose data $\hat{D}^t_{\text{tgt}} = \text{Dec}(S_{\text{tgt}}, z^t)$ based on the target skeleton. Decoder output $\hat{D}^t_{\text{tgt}} = \{q^t_{\text{tgt}}, r^t_{\text{tgt}}, c^t_{\text{tgt}}\}$ is represented as a subset of D^t and omit redundant elements such as p^t, p^{t-1} and v^t. Similar to the encoder, the decoder comprises multiple graph convolution layers. We concatenate z^t to each target skeleton joint $S_{1:J}$ for the first graph convolution layer. From the second layer, output from the previous GCN layer is concatenated to $S_{1:J}$. Since we want our final result $\hat{D}^t_{\text{tgt}}$ in a graph form, graph pooling is not applied to the decoder.

4.3 Training

4.3.1 Data Preparation. To train our autoencoder for skeleton-agnostic motion embedding, we start by creating a motion database $\mathcal{M} = \{M_1, M_2, \cdots, M_N\}$ by collecting diverse skeletal motions available online. Each motion clip $M = (S, D^{1:T})$ may have a different skeleton from the other clips, as they were collected from various sources. Furthermore, we build a skeleton database $\mathcal{S} = \{S_1, S_2, \cdots, S_K\}$ where K represents the number of distinct skeletons we acquired. Then we apply augmentation techniques to both databases to enhance training effectiveness. While the detailed pseudo-codes for augmentation are provided in Appendix, here we outline the basic principles of the augmentation process.

Our skeleton augmentation algorithm applied to a skeleton database $\mathcal{S}$, generates an expanded database $\mathcal{S}'$ with a wider range of diverse skeletons. The algorithm randomly selects a skeleton S from the database and introduces variations based on our observations. These variations include random creation or removal of spine, neck, and shoulder joints and random scaling of limb lengths and root height. Furthermore, we observed that some characters have dummy joints near the hips and end-effectors. We also randomly add dummy joints to account for this characteristic. The process of generating random skeletons is repeated $H > K$ times, resulting in the augmented skeleton database $\mathcal{S}' = \{S'_1, S'_2, \cdots, S'_H\}$, where H represents the size of the augmented database.

The motion database augmentation takes the original motion database $\mathcal{M}$ and augmented skeleton database $\mathcal{S}'$ as input, resulting in the augmented motion database $\mathcal{M}'$. For each motion $M_i = (S_i, D_i^{1:T})$ in $\mathcal{M}$, we randomly select target skeletons from $\mathcal{S}'$. We then use an off-the-shelf retargeting tool in MotionBuilder [Autodesk 2021] to retarget M_i with the target skeletons, where we use the default parameters provided by the software. The resulting augmented database contains new motion pairs with the same semantics but realized by different skeletons. Note that the retargeting here is an offline procedure, converting one skeletal motion to another skeletal motion only, while our autoencoder essentially learns retargeting for the entire skeletons.

4.3.2 Loss. Given an augmented motion database, we randomly select a source motion $M_{\text{src}} = (S_{\text{src}}, D^{1:T}_{\text{src}})$ and a target motion $M_{\text{tgt}} = (S_{\text{tgt}}, D^{1:T}_{\text{tgt}})$ that correspond to the same semantic motion. We then feed them to the autoencoder, which produces an output motion denoted as $\hat{D}^{1:T}_{\text{src}\to\text{tgt}}$ using the equation:

$$\hat{D}^{1:T}_{\text{src}\to\text{tgt}} = \text{Dec}(S_{\text{tgt}}, \text{Enc}(S_{\text{src}}, D^{1:T}_{\text{src}})). \tag{3}$$

where the sequential feeding of pose data to the encoder and the decoder are denoted as $\text{Enc}(S, D^{1:T})$ and $\text{Dec}(S, z^{1:T})$ respectively for brevity. In this autoencoding process, if the source and target skeletons are the same, the output motion should be identical to the input motion. On the other hand, if they are different, the output motion $\hat{D}^{1:T}_{\text{src}\to\text{tgt}}$ should be the same as the target motion $D^{1:T}_{\text{tgt}}$. Note that both cases can be processed in a unified manner by comparing the model output $\hat{D}^{1:T}_{\text{src}\to\text{tgt}}$ with the target motion $D^{1:T}_{\text{tgt}}$ only. Although motion reconstruction is the most crucial condition that the autoencoder should satisfy, training solely based on this condition often leads to local minima that do not produce visually satisfactory reconstruction results. Therefore, we incorporate auxiliary loss terms into our training process. The loss function used to train our model is defined as follows:

$$L = \sum_{t=1,\cdots,F} L^t_{\text{rec}} + L^t_{\text{vel}} + L^t_{\text{con}} + L^t_z, \tag{4}$$

where F is the total number of frames in the training batch. Our loss function consists of four terms: the reconstruction loss, the velocity loss, the contact label loss, and the embedding consistency loss.

Reconstruction Loss. The reconstruction loss, as the name implies, is a term that ensures the reconstruction condition. It compares the output motion $\hat{D}^{1:T}_{\text{src}\to\text{tgt}}$ with the ground truth motion $D^{1:T}_{\text{tgt}}$.

$$\begin{aligned} L^t_{\text{rec}} = 5\|q^t_{\text{tgt}} - \hat{q}^t_{\text{src}\to\text{tgt}}\|^2 + 0.01\|p^t_{\text{tgt}} - FK(\hat{q}^t_{\text{src}\to\text{tgt}})\|^2 \\ +10\|r^t_{\text{tgt}} - \hat{r}^t_{\text{src}\to\text{tgt}}\|^2 \end{aligned} \tag{5}$$

where reconstruction quality is evaluated by errors in joint rotation q, joint position p, and root joint movement r. To compute joint positions from joint rotations, we implement the forward kinematics layer called FK. Implementing FK for our framework is challenging because it should be able to adapt to multiple skeletons automatically, whereas FK in other methods was a fixed mapping [Pavllo et al. 2018]. To address this, we compute FK for joints of the same depth in a batch, sequentially from zero-depth joints (root joints) to the maximum-depth joints.

Temporal Coherence Loss. Our autoencoder operates on a per-frame basis, allowing it to process motions of different lengths in

the same manner. To ensure temporal coherence between adjacent frames, we incorporate a velocity loss which is computed as follows:

$$\begin{aligned} L^t_{\text{vel}} &= \|v^t - \hat{v}^t\|^2 \\ v^t &= (p^t - p^{t-1})/\Delta t \\ \hat{v}^t &= (FK(\hat{q}^t_{\text{src}\to\text{tgt}}) - FK(\hat{q}^{t-1}_{\text{src}\to\text{tgt}}))/\Delta t \end{aligned} \tag{6}$$

where we compare the output joint linear velocity with the ground truth joint linear velocity [Tevet et al. 2023]. In addition, to penalize excessive acceleration change, we use jerk loss which is computed as follows:

$$\begin{aligned} L^t_{\text{jerk}} &= \|(\hat{a}^t - \hat{a}^{t-1})/\Delta t\|^2 \\ \hat{a}^t &= (\hat{v}^t - \hat{v}^{t-1})/\Delta t \end{aligned} \tag{7}$$

Contact Loss. Foot sliding is a common artifact encountered in deep learning-based motion generation, arising from the smoothing effect of the network output. Handling this artifact is crucial as our autoencoder is expected to perform retargeting between characters with highly different skeletons. To address this, we introduce a contact loss similar to [Harvey et al. 2020a; Holden et al. 2017; Lee et al. 2018; Shi et al. 2020; Tevet et al. 2023], which is defined as follows:

$$L^t_{\text{con}} = \|c^t - \hat{c}^t\|^2 + c^t\|\hat{v}^t\|^2 \tag{8}$$

$$+\|\text{clamp}(1 - \frac{\hat{y}^t}{y_{\text{thres}}}, 0, 1) \cdot \hat{v}^t\|^2 + \|\min(\hat{y}^t, 0)\|^2 \tag{9}$$

The first term penalizes the discrepancy between the ground truth contact label c^t and the predicted contact label $\hat{c}^t$ The second term aims to minimize joint linear velocity when contact is activated. the joint position is close to the ground, where we check if its height is less than a threshold y_{thres} (3cm in our implementation). The last term prevents joint positions from penetrating the ground. The contact loss is computed for all joints.

Embedding Consistency Loss. The objective of the encoder is to learn a skeleton-agnostic motion embedding, where motions with different skeletons but identical semantics should be mapped to the same embedding. This condition can be expressed as follows [Aberman et al. 2020]:

$$L^t_z = \|\text{Enc}(S_{src}, D^t_{src}) - \text{Enc}(S_{tgt}, D^t_{tgt}))\|^2. \tag{10}$$

where we can directly measure the difference of the embeddings computed from two input motions $(S_{\text{src}}, D^t_{\text{src}})$, $(S_{\text{tgt}}, D^t_{\text{tgt}})$ that represent identical semantics but are realized by different skeletons $(S_{\text{src}} \neq S_{\text{tgt}})$.

5 SKELETON-AGNOSTIC ANIMATION TASKS

In this section, we will describe how we address general animation tasks in a skeleton-agnostic manner using the autoencoder we have built.

5.1 Motion Classification

Motion classification aims to assign a category label to the input motion. It can be represented as a mapping $T : M \to h$, where M is the input motion and h is the output label. Using the encoder of our autoencoder we've built, we learn a mapping $T : z \to h$ from the SAME space to the output label. This way, we can classify motions with different skeletons using a unified model.

5.2 Retargeting

Retargeting is an animation task that involves transforming motions from a source character to motions suitable for a target character, typically with different skeletal structures. In our framework, the autoencoding process described in Equation 3 is essentially equivalent to the retargeting task when the source and target skeletons differ.

5.3 Reconstruction of Missing Joints

During the retargeting process, if the target character has extra links that the source character does not have, traditional algorithms typically solve this by transforming the motions of joints only that have corresponding counterparts while the motions of other joints remain fixed. However, our framework offers the capability to reconstruct new motions for extra joints. When selecting a motion pair $(S_{\text{src}}, D^{1:T}_{\text{src}})$ and $(S_{\text{tgt}}, D^{1:T}_{\text{tgt}})$ from the training motion database, we randomly eliminate an entire limb or its partial links. Joints are randomly selected and all the links between the selected joint and its end-effector are removed from the skeleton. In the GCN context, this can be implemented by masking the joints and edges designated for deletion while excluding their corresponding loss terms as well. We apply this elimination process to both the source and target skeletons. We can also fine-tune a model to focus on specific joint reconstruction. We demonstrate two scenarios for missing joints reconstruction:

- Full-body motion reconstruction from upper-body motion.
- Finger motion reconstruction from full-body motion.

5.4 Motion Similarity

Assessing the similarity between two input motions is a fundamental animation task that has applications in various areas, such as searching for similar motions in large databases and constructing motion graphs that connect similar frames. Traditionally, similarity metrics defined in the motion space, such as comparing joint angles and velocities, have been widely used, and other sophisticated metrics have also been proposed [Aristidou et al. 2018; Müller and Röder 2006]. Our framework can also assess the similarity between two input motions $M_a = (S_a, D^{1:T_a}_a)$, $M_b = (S_b, D^{1:T_b}_b)$ in the latent space:

$$\begin{aligned} z^{t_a}_a, z^{t_b}_b &= \text{Enc}(S_a, D^{t_a}_a), \text{Enc}(S_b, D^{t_b}_b) \\ d_p(x, y) &= \|x - y\|^2 \\ d_m &= \text{DTW}(z^{1:T_a}_a, z^{1:T_b}_b, d_p(\cdot, \cdot)) \end{aligned} \tag{11}$$

where dynamic time warping (DTW) [Müller 2007] is used to handle the difference in motion length.

5.5 Interactive Character Controllers

Interactive character controllers generate responsive motions based on user control, commonly used in applications like computer games. These controllers can be represented as a mapping function T :

$D^t, u^t \rightarrow D_{t+1}$, where the current pose D^t and user control u^t are used to compute the next pose D^{t+1}. Motion matching is a popular technique for implementing this mapping which involves searching for the best next pose D^{t+1} in a motion database $\{(D_i, C_i)_{i=1,2,\dots}\}$. C_i is an additional annotation of data D_i, such as motion category, often precomputed and stored along with the data. Search cost is described as a weighted sum of two terms, $\phi_1(D^t, D_i)+w\cdot\phi_2(D_i, C_i)$, where ϕ_1, ϕ_2, and w represent a transition cost from the current pose D^t to pose D_i in the database, user control satisfaction cost, and relative weight between them, respectively.

Our framework enables motion matching with heterogenous motion database. We first construct a new motion matching database $\{(z_i, C_i)_{i=1,2,\dots}\}$ by converting the original database into the SAME space, where z_i represents the pose embeddings. During runtime, we compute ϕ_1 by

$$\phi_1^{SAME}(D^t, z_i) = \|\text{Enc}(S, D^t) - z_i\|^2, \tag{12}$$

where the current pose D^t of the character S is first projected into the SAME space to measure squared $L2$ distance, by which we find the nearest neighbors. Once we find out the best embedding z_* minimizing both ϕ_1 and ϕ_2, it is then realized back to the original motion space via the decoding process $\text{Dec}(S, z_*)$. This process repeats in an auto-regressive manner. By using only a single motion matching module implemented using our framework for various characters, it can reduce huge memory consumption. Moreover, on-the-fly morphing to other characters is even possible by changing the target skeleton during decoding.

6 EXPERIMENTS

6.1 Implementation Detail

Our framework is implemented primarily in Python, utilizing the PyTorch library [Paszke et al. 2019] for the deep learning components. For graph-related operations, we utilize the PyTorch-Geometric library [Fey and Lenssen 2019]. The encoder and decoder modules in our framework are built using multi-headed graph attention layers. For a more detailed description of the network architecture, please refer to Appendix. During training, we employ a minibatch size of 2048 (256 samples of 8 consequent frames) and utilize the Adam optimizer with an initial learning rate of 1e-2 and decreasing exponentially down to 1e-4 with $\gamma = 0.99$. While the detailed pseudo-codes for augmentation are provided in Appendix, here we outline the basic principles of the augmentation process.

6.2 Training Data Preparation

To train a base autoencoder model, we prepared $\mathcal{S}_{\text{train}}$ of 92 character variations and $\mathcal{M}_{\text{train}}$ comprising 130 minutes of motions from various sources including Lafan1 [Harvey et al. 2020b], ACCAD [OSU [n.d.]], TotalCapture [Trumble et al. 2017], PFNN [Holden et al. 2017], and SFU dataset [SFU 2011]. We excluded motions involving object interactions, such as climbing stairs or lying on a bed. For more detail of the dataset composition, please refer to Appendix Section A. Next, we constructed $\mathcal{S}'_{\text{train}}$ through the skeleton augmentation algorithm, and $\mathcal{M}'_{\text{train}}$ by applying the motion augmentation algorithm with $\mathcal{S}'_{\text{train}}$ and $\mathcal{M}_{\text{train}}$. Augmented datasets comprise 780 minutes of motions and 160 types of skeletons.

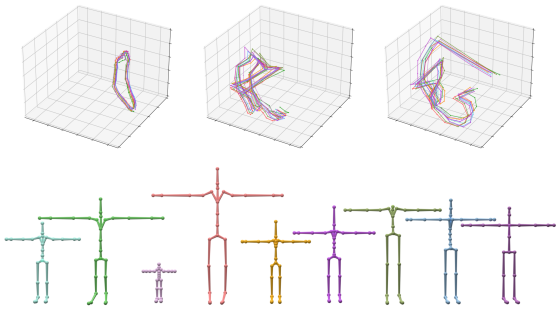

Fig. 4. Embeddings of 3 different motions (top-left: walk, top-middle: jump-rope, top-right: tennis) for nine different skeletons (bottom). The embedding are created by our autoencoder then projected into 3D space via Principle Component Analysis.

6.3 Embedding Space Analysis

6.3.1 Visualization. Figure 4 shows the embeddings of motions with identical semantics but different skeletons, which were projected into a 2D space using Principle Component Analysis (sse Figure 4). We conducted tests using three motion clips (walk, jump-rope, and tennis) with 9 different skeletons. The result shows that our autoencoder produces highly similar embeddings for motions that share the same semantics, which implies that our SAME space effectively captures the semantics of input motions well.

6.3.2 Arithmetic Operations. We have observed that arithmetic operations such as addition, subtraction, or multiplication on our motion embedding often yield semantically meaningful and valid results, even though such constraints were not explicitly enforced during training. Let's say we have three motions $M_a^{1:T}$, $M_b^{1:T}$, and $M_c^{1:T}$ representing a standing motion with hand waving, a standing motion, and a walking motion, respectively, and $z_a^{1:T}$, $z_b^{1:T}$, and $z_c^{1:T}$ are their embeddings computed by our method. Figure 10 showcases examples of such operations, where $z_a^{1:T} - z_b^{1:T} + z_c^{1:T}$ generates a walking motion with hand waving, and $0.5 \cdot (z_a^{1:T} - z_b^{1:T}) + z_c^{1:T}$ generates a walking motion with a smaller hand waving. These results demonstrate the potential of our framework to perform motion composition and manipulation in a meaningful and intuitive manner.

Table 1. Component Ablation (Autoencoder Reconstruction Performance)

	JR [rad]	RT [cm]	JP [cm]	FS	GP [cm]
Ours	0.2769	**1.2846**	5.2018	**0.0075**	**0.0034**
w/o contact loss	**0.2534**	1.4482	**4.5685**	0.1056	0.1345
w/o velocity loss	0.2734	1.5028	6.0059	0.0098	0.0037
w/o random joint masking	0.2756	1.7324	5.7918	0.0291	0.0248
$\mathcal{S}'$ by one skeleton	0.3073	2.204	9.1537	0.0283	0.0367

6.4 Evaluation of Reconstruction on Unseen Data

We evaluated the generalization capability of our learned autoencoder by measuring the reconstruction performance on unseen data that comprises 11 skeletal variations and 67 minutes of motions from Mixamo [Adobe 2021], MotionBuilder [Autodesk 2021] and SNU Actor database [SNU 2013]. Please refer to Appendix Table 3 for the detailed composition. The first row in Table 1 shows the average errors in reconstruction for the motions in the test dataset. The errors were measured using five metrics: joint rotation (JR) $\sum_t \sum_j \|log(R(\hat{q}_j^t)^{-1} R(q_j^t))\|$, joint position (JP) $\sum_t \sum_j \|\hat{p}_j^t - p_j^t\|$, root trajectory (RT) $\sum_t \|\hat{p}_{\text{root}}^t - p_{\text{root}}^t\|$, foot skating (FS) [Zhang and van de Panne 2018], and ground penetration (GP) $\sum_t \sum_j \hat{d}_j$, where R is 3x3 rotation matrix constructed by 6D representation, log projects R to axis-angle representation, $\hat{p}_{\text{root}}^t$ is the root joint position projected on the ground, $\hat{d}_j$ is the penetration depth of j-th joint to the ground. When computing the metrics, all motion clips were first split into several one-second=long motions to account for variations in duration. The result demonstrates that our learned autoencoder achieves successful reconstruction even for unseen motions with unseen skeletons, yielding small errors.

Table 2. Performance comparisons on the retargeting task.

(unit: 10^3)

Method	Intra	Cross
[Villegas et al. 2018]	6.24	243
[Aberman et al. 2020] (no L_{adv})	0.47	3.81
[Aberman et al. 2020] (full approach)	2.76	2.25
Ours	2.91	2.47

6.5 Animation Tasks

We showcase the results of the animation tasks introduced in Section 5, which we solve in a skeleton-agnostic manner using our framework. The results are best seen in the supplemental video. Note that we ultimately aim to develop a base model capable of performing various downstream tasks using the specific dataset of interest, which may not be included during the base model's training process. Therefore, for animation tasks, we utilized dataset that were not used during autoencoder training to simulate such scenario (Appendix Table 3).

6.5.1 Motion Classification. We trained a multi-label classifier using a Transformer [Vaswani et al. 2017] consisting of a single-layer encoder and decoder. For training, we gathered 276 motion clips of 40 subjects from the CMU motion database [CMU 2006], belonging to 30 different categories. From this dataset, we randomly selected 1-4 clips per category to create a validation set of 60 clips. The remaining 216 clips formed the training set as a result. We employed the categorical cross-entropy loss function and the Adam optimizer with a minibatch size of 48. Our classifier achieved an accuracy of 95% (57/60) on the validation set. This result highlights the potential of our embedding as valuable features for classification tasks.

6.5.2 Retargeting. Figure 9 showcases the retargeted motions to 6 characters from a single input motion, whose skeletons are significantly different from each other. Source motions and target characters are from a separate database and unseen during training.

6.5.3 Motion Similarity. Given motions from Mixamo dataset, we search for the most similar motions from MotionBuilder database, which has different skeleton configuration. Figure 7 illustrates examples of search results, including squat, raising arms, and boxing uppercut.

6.5.4 Missing Joint Reconstruction. First, our method succeeds to reconstruct high-quality full-body motions from the upper-body motions only for various scenarios including walking, running, squat, kicking, and boxing (see Figure 6). We fine-tuned the encoder of the base model by masking the lower body part to the encoder. Second, our method can also synthesize extra finger motions given full-body motions without fingers. Using Mixamo dataset with finger motion, we fine-tuned the decoder by feeding motion without fingers to the encoder and a target skeleton with finger to the decoder. We tested generating finger motions for MotionBuilder dataset which does not have ground truth finger motion and is unseen during training (see Figure 8).

6.5.5 Interactive Character Control via Motion Matching. We implement a motion matching controller using a database consisting of three distinctive types of motions: locomotion from LAFAN1, crouching from PFNN, and kicking from SNU ActorDB. Figure 1 demonstrates how our motion matching controller enables users to generate motions seamlessly transitioning among these heterogeneous datasets and even allows for character morphing on the fly.

6.6 Ablation Study

To verify the effectiveness of the design choices made during the development of our framework, we conducted ablation study, measuring the same five metrics used in Section 6.4. The results are presented in Table 1. Our method, including all the components, clearly outperforms all the variations where one of the components is missing.

6.7 Comparison with Other Methods

Table 2 shows comparisons of retargeting performance with two previous state-of-the-art retargeting methods [Aberman et al. 2020] and [Villegas et al. 2018]. We computed the average error normalized by the character height using the test data employed in the two earlier studies. Among the test data consisting of 106 motion clips, we excluded 11 clips captured with objects (e.g., bar-hanging) because such object-interaction motions were completely excluded during the training of our model. The result shows that our method using only a single network demonstrates comparable performance to the previous methods specialized for the retargeting task.

7 DISCUSSION

We present a framework for solving animation tasks in a skeleton-agnostic manner. Our approach utilizes a novel autoencoder based

on a graph convolution network. The autoencoder learns an embedding space called Skeleton-Agnostic Motion Embedding (SAME), which efficiently captures the semantics of motion while disregarding skeleton-specific information during the encoding process. This enables us to perform general animation tasks, such as motion classification, retargeting, and interactive character control, in a skeleton-agnostic manner. The processed embeddings can then be used to generate motions with arbitrary skeletons based on the target applications.

While our framework has shown success in solving animation tasks in a skeleton-agnostic manner, there are some limitations to consider. Firstly, training our autoencoder requires motion pairs with identical semantics but distinct skeletons. Although we've shown the effectiveness of our proposed semi-automatic augmentation algorithm for biped characters, it is not straightforward to expand beyond human-like characters. Secondly, like other deep learning-based methods, the quality of generated motions and the ability to process unseen inputs are constrained by the training data. If the character has functional joints (e.g. tails, ornaments) that are unseen during training or has a completely different t-pose, the result quality degrades. Training our autoencoder in an unsupervised way or expanding our framework to handle a broader range of subjects are interesting future research directions.

We believe that our approach has the potential to pave the way for the development of a large-scale motion model by enabling the integration and utilization of numerous datasets, not only those currently available but also future ones. This would contribute to the advancement of the field of computer animation and related research areas.

ACKNOWLEDGMENTS

We thank Jiye Lee and Sehee Min for their efforts in refining the initial draft of the manuscript and video editing. This work was supported by NCSOFT AI Center under Project Number NCSOFT-2023-1283-K. Jungdam Won and Sunmin Lee were partially supported by the New Faculty Startup Fund from Seoul National University, ICT(Institute of Computer Technology) at Seoul National University, and the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2023-2020-0-01460) supervised by the IITP(Institute for Information & Communications Technology Planning & Evaluation).

REFERENCES

Kfir Aberman, Peizhuo Li, Dani Lischinski, Olga Sorkine-Hornung, Daniel Cohen-Or, and Baoquan Chen. 2020. Skeleton-aware networks for deep motion retargeting. *ACM Transactions on Graphics (TOG)* 39, 4 (2020), 62–1.

Kfir Aberman, Rundi Wu, Dani Lischinski, Baoquan Chen, and Daniel Cohen-Or. 2019. Learning character-agnostic motion for motion retargeting in 2d. *arXiv preprint arXiv:1905.01680* (2019).

Adobe. 2021. *Mixamo Animation Dataset.* https://www.mixamo.com/

Andreas Aristidou, Daniel Cohen-Or, Jessica K Hodgins, Yiorgos Chrysanthou, and Ariel Shamir. 2018. Deep motifs and motion signatures. *ACM Transactions on Graphics (TOG)* 37, 6 (2018), 1–13.

Autodesk. 2021. *Motion Builder - a 3D character animation software.* Autodesk. https://www.autodesk.com/

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs.CL]

Jinxiang Chai and Jessica K Hodgins. 2005. Performance animation from low-dimensional control signals. In *ACM SIGGRAPH 2005 Papers*. 686–696.

Kwang-Jin Choi and Hyeong-Seok Ko. 1999. On-line motion retargetting. In *Proceedings of Pacific Graphics*. 32–42.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. PaLM: Scaling Language Modeling with Pathways. arXiv:2204.02311 [cs.CL]

CMU. 2006. *CMU Graphics Lab Motion Capture Database.* http://mocap.cs.cmu.edu/

Matthias Fey and Jan E. Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Saeed Ghorbani, Calden Wloka, Ali Etemad, Marcus A Brubaker, and Nikolaus F Troje. 2020. Probabilistic character motion synthesis using a hierarchical deep latent variable model. In *Computer Graphics Forum*, Vol. 39. 225–239.

Michael Gleicher. 1998. Retargetting motion to new characters. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques (I3D)*. 33–42.

Ikhsanul Habibie, Daniel Holden, Jonathan Schwarz, Joe Yearsley, and Taku Komura. 2017. A recurrent variational autoencoder for human motion synthesis. In *Proceedings of British Machine Vision Conference (BMVC)*.

Félix G. Harvey, Mike Yurick, Derek Nowrouzezahrai, and Christopher Pal. 2020a. Robust Motion In-Betweening. *ACM Trans. Graph.* 39, 4, Article 60 (aug 2020), 12 pages. https://doi.org/10.1145/3386569.3392480

Félix G. Harvey, Mike Yurick, Derek Nowrouzezahrai, and Christopher Pal. 2020b. Robust Motion In-Betweening. *ACM Trans. Graph.* 39, 4 (2020).

Chengan He, Jun Saito, James Zachary, Holly Rushmeier, and Yi Zhou. 2022. Nemf: Neural motion fields for kinematic animation. *arXiv preprint arXiv:2206.03287* (2022).

Daniel Holden, Taku Komura, and Jun Saito. 2017. Phase-functioned neural networks for character control. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–13.

Daniel Holden, Jun Saito, Taku Komura, and Thomas Joyce. 2015. Learning motion manifolds with convolutional autoencoders. In *SIGGRAPH Asia 2015 technical briefs*. 1–4.

Shuaiying Hou, Weiwei Xu, Jinxiang Chai, Congyi Wang, Wenlin Zhuang, Yu Chen, Hujun Bao, and Yangang Wang. 2021. A Causal Convolutional Neural Network for Motion Modeling and Synthesis. *arXiv preprint arXiv:2101.12276* (2021).

Linjiang Huang, Yan Huang, Wanli Ouyang, and Liang Wang. 2020. Part-level graph convolutional network for skeleton-based action recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 11045–11052.

Deok-Kyeong Jang, Soomin Park, and Sung-Hee Lee. 2022. Motion Puzzle: Arbitrary Motion Style Transfer by Body Part. *arXiv preprint arXiv:2202.05274* (2022).

Jehee Lee, Jinxiang Chai, Paul SA Reitsma, Jessica K Hodgins, and Nancy S Pollard. 2002. Interactive control of avatars animated with human motion data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. 491–500.

Jehee Lee and Sung Yong Shin. 1999. A hierarchical approach to interactive motion editing for human-like figures. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques (I3D)*. 39–48.

Kyungho Lee, Seyoung Lee, and Jehee Lee. 2018. Interactive Character Animation by Learning Multi-Objective Control. 37, 6, Article 180 (dec 2018), 10 pages. https://doi.org/10.1145/3272127.3275071

Jiaman Li, Ruben Villegas, Duygu Ceylan, Jimei Yang, Zhengfei Kuang, Hao Li, and Yajie Zhao. 2021. Task-generic hierarchical human motion prior using vaes. In *2021 International Conference on 3D Vision (3DV)*. 771–781.

Maosen Li, Siheng Chen, Xu Chen, Ya Zhang, Yanfeng Wang, and Qi Tian. 2019. Actional-structural graph convolutional networks for skeleton-based action recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*. 3595–3603.

Jongin Lim, Hyung Jin Chang, and Jin Young Choi. 2019. PMnet: Learning of Disentangled Pose and Movement for Unsupervised Motion Retargeting.. In *Proceedings of British Machine Vision Conference (BMVC)*, Vol. 2. 7.

Hung Yu Ling, Fabio Zinno, George Cheng, and Michiel van de Panne. 2020. Character Controllers Using Motion VAEs. *ACM Trans. Graph.* 39, 4 (2020).

Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. 2015. SMPL: A Skinned Multi-Person Linear Model. *ACM Trans. Graph.* 34, 6 (Oct. 2015), 248:1–248:16.

Naureen Mahmood, Nima Ghorbani, Nikolaus F Troje, Gerard Pons-Moll, and Michael J Black. 2019. AMASS: Archive of motion capture as surface shapes. In *Proceedings of the IEEE/CVF international conference on computer vision (CVPR)*. 5442–5451.

Meinard Müller. 2007. Dynamic time warping. *Information retrieval for music and motion* (2007), 69–84.

Meinard Müller and Tido Röder. 2006. Motion templates for automatic classification and retrieval of motion capture data. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 137–146.

OSU. [n.d.]. *OSU ACCAD*. https://accad.osu.edu/

Jungnam Park, Sehee Min, Phil Sik Chang, Jaedong Lee, Moonseok Park, and Jehee Lee. 2022. Generative GaitNet. *arXiv preprint arXiv:2201.12044* (2022).

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*. 8024–8035.

Georgios Pavlakos, Vasileios Choutas, Nima Ghorbani, Timo Bolkart, Ahmed A. A. Osman, Dimitrios Tzionas, and Michael J. Black. 2019. Expressive Body Capture: 3D Hands, Face, and Body from a Single Image. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. 10975–10985.

Dario Pavllo, David Grangier, and Michael Auli. 2018. Quaternet: A quaternion-based recurrent model for human motion. *arXiv preprint arXiv:1805.06485* (2018).

Sigal Raab, Inbal Leibovitch, Peizhuo Li, Kfir Aberman, Olga Sorkine-Hornung, and Daniel Cohen-Or. 2023. Modi: Unconditional motion synthesis from diverse data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 13873–13883.

Davis Rempe, Tolga Birdal, Aaron Hertzmann, Jimei Yang, Srinath Sridhar, and Leonidas J Guibas. 2021. Humor: 3d human motion model for robust pose estimation. In *Proceedings of the IEEE/CVF international conference on computer vision*. 11488–11499.

Alla Safonova, Jessica K Hodgins, and Nancy S Pollard. 2004. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Transactions on Graphics (ToG)* 23, 3 (2004), 514–521.

SFU. 2011. *SFU Motion Capture Database*. https://mocap.cs.sfu.ca/

Lei Shi, Yifan Zhang, Jian Cheng, and Hanqing Lu. 2019. Two-stream adaptive graph convolutional networks for skeleton-based action recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (CVPR)*. 12026–12035.

Mingyi Shi, Kfir Aberman, Andreas Aristidou, Taku Komura, Dani Lischinski, Daniel Cohen-Or, and Baoquan Chen. 2020. MotioNet: 3D Human Motion Reconstruction from Monocular Video with Skeleton Consistency. *arXiv preprint arXiv:2006.12075* (2020).

Hyun Joon Shin and Jehee Lee. 2006. Motion synthesis and editing in low-dimensional spaces. *Computer Animation and Virtual Worlds (CASA 2006)* 17 (2006), 219–227. Issue 3-4.

Hyun Joon Shin, Jehee Lee, Sung Yong Shin, and Michael Gleicher. 2001. Computer puppetry: An importance-based approach. *ACM Transactions on Graphics (TOG)* 20, 2 (2001), 67–94.

SNU. 2013. *SNU Motion Database*. https://mrl.snu.ac.kr/mdb/

Sebastian Starke, Ian Mason, and Taku Komura. 2022. DeepPhase: Periodic Autoencoders for Learning Motion Phase Manifolds. *ACM Trans. Graph.* 41, 4, Article 136 (jul 2022), 13 pages. https://doi.org/10.1145/3528223.3530178

David J Sturman. 1998. Computer puppetry. *IEEE Computer Graphics and Applications* 18, 1 (1998), 38–45.

Seyoon Tak and Hyeong-Seok Ko. 2005. A physically-based motion retargeting filter. *ACM Transactions on Graphics (TOG)* 24, 1 (2005), 98–117.

Guy Tevet, Sigal Raab, Brian Gordon, Yoni Shafir, Daniel Cohen-or, and Amit Haim Bermano. 2023. Human Motion Diffusion Model. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=SJ1kSyO2jwu

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL]

Matt Trumble, Andrew Gilbert, Charles Malleson, Adrian Hilton, and John Collomosse. 2017. Total Capture: 3D Human Pose Estimation Fusing Video and Inertial Sensors. In *2017 British Machine Vision Conference (BMVC)*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

Ruben Villegas, Jimei Yang, Duygu Ceylan, and Honglak Lee. 2018. Neural kinematic networks for unsupervised motion retargetting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 8639–8648.

Jack M Wang, David J Fleet, and Aaron Hertzmann. 2007. Gaussian process dynamical models for human motion. *IEEE transactions on pattern analysis and machine intelligence* 30, 2 (2007), 283–298.

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. 2019. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)* 38, 5 (2019), 1–12.

Jungdam Won and Jehee Lee. 2019. Learning body shape variation in physics-based characters. *ACM Transactions on Graphics (TOG)* 38, 6 (2019), 1–12.

Zhan Xu, Yang Zhou, Evangelos Kalogerakis, Chris Landreth, and Karan Singh. 2020. Rignet: Neural rigging for articulated characters. *arXiv preprint arXiv:2005.00559* (2020).

Sijie Yan, Yuanjun Xiong, and Dahua Lin. 2018. Spatial temporal graph convolutional networks for skeleton-based action recognition. In *Thirty-second AAAI conference on artificial intelligence*.

Ye Yuan, Jiaming Song, Umar Iqbal, Arash Vahdat, and Jan Kautz. 2022. PhysDiff: Physics-Guided Human Motion Diffusion Model. *arXiv preprint arXiv:2212.02500* (2022).

Xinyi Zhang and Michiel van de Panne. 2018. Data-driven autocompletion for keyframe animation. In *Proceedings of the 11th ACM SIGGRAPH Conference on Motion, Interaction and Games*. 1–11.

Yi Zhou, Connelly Barnes, Lu Jingwan, Yang Jimei, and Li Hao. 2019. On the Continuity of Rotation Representations in Neural Networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

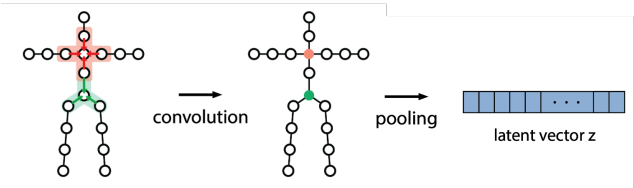

Fig. 5. Network operations used in our model. Graph convolution update the graph node (joint) feature by aggregating its own and neighbors' features. Graph pooling generates a fixed-length vector by summarizing all node features.

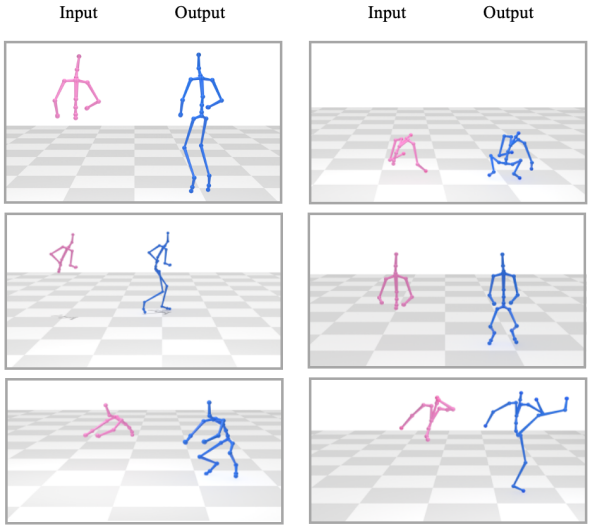

Fig. 6. Full-body reconstruction from the upper body only.

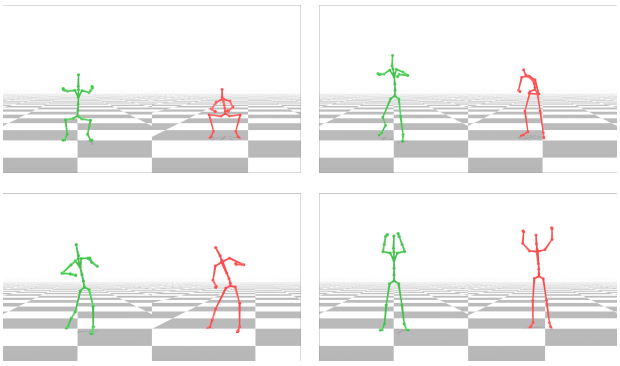

Fig. 7. Motion similarity. The left (green) is a search key motion from Mixamo dataset, and the right (red) is the most similar motion searched from the MotionBuilder dataset. We assess the similarity between two input motions of different skeletons by measuring distance in the SAME space. Dynamic time warping is used to handle the difference in motion length.

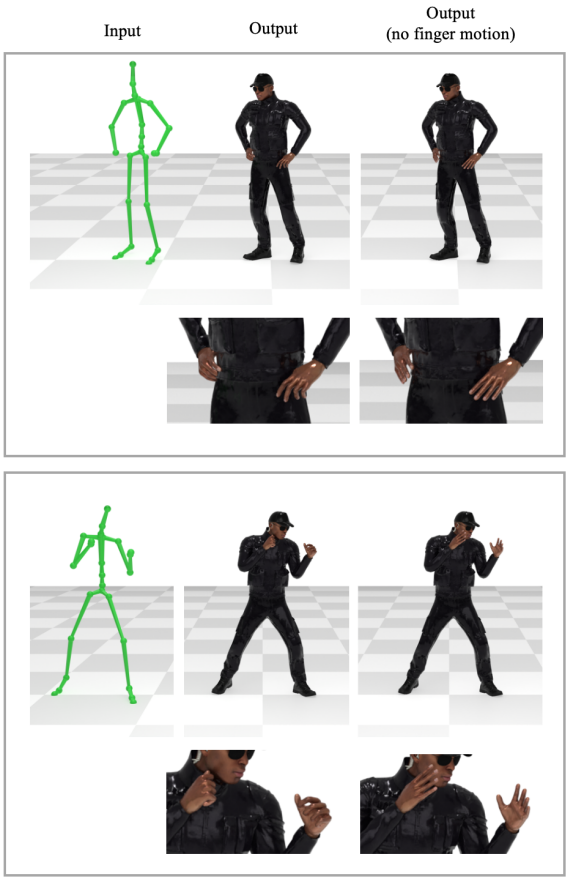

Fig. 8. Finger motion reconstruction from the full body without finger motions.

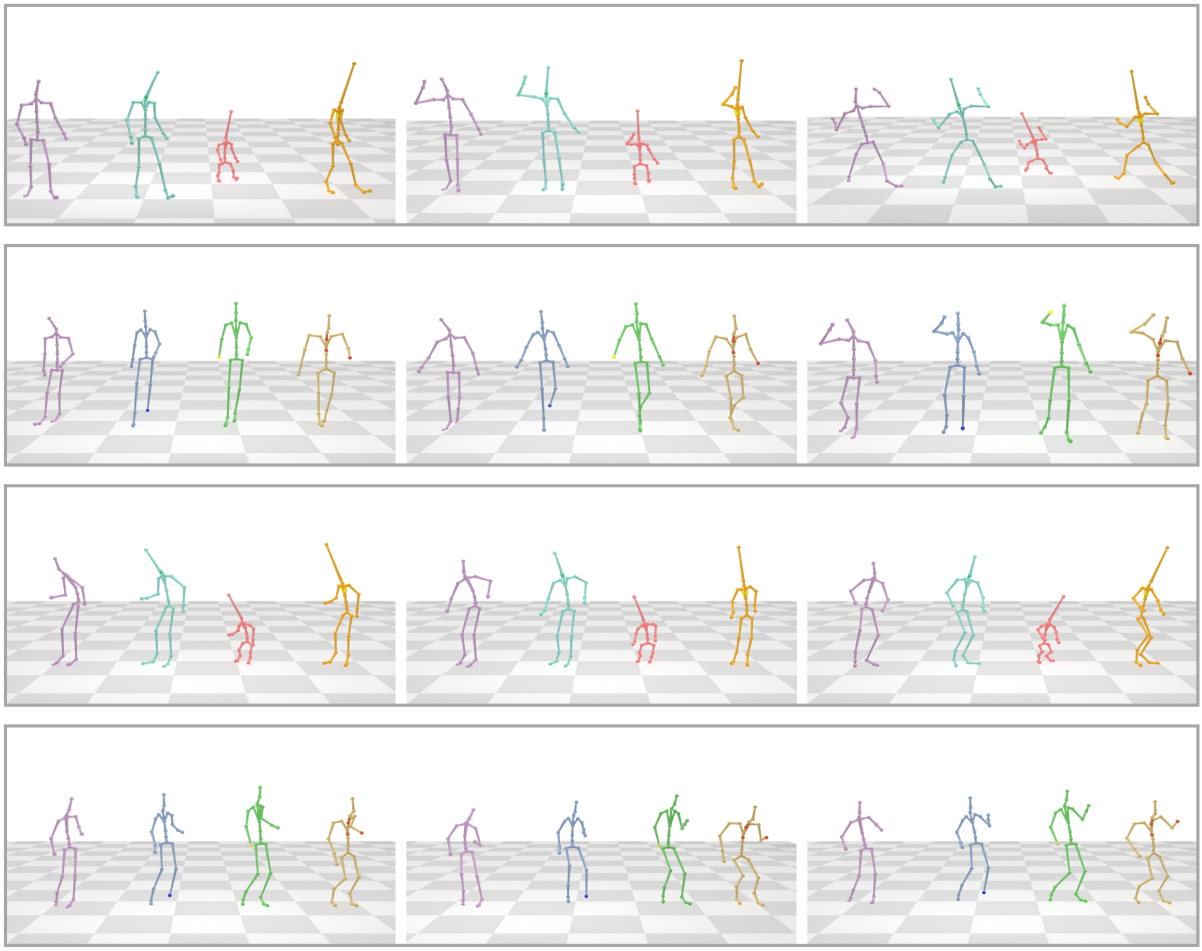

Fig. 9. Retargeting. The left most character (purple) is the source and the remaining characters are retargeted.

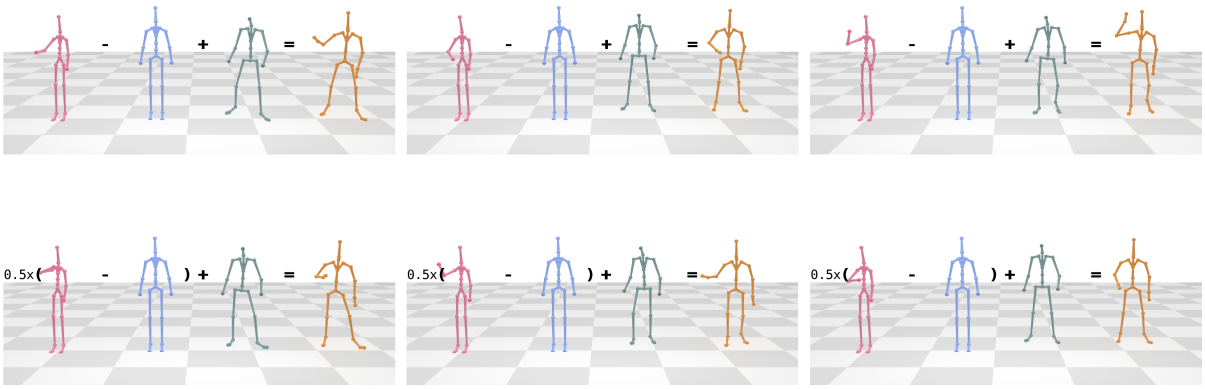

Fig. 10. Arithmetic operations in the SAME space. The top figure represents "Standing motion with hand waving (pink)" minus "standing motion (blue)" plus "walking motion (green)", and the bottom figure represents 0.5 times ["Standing motion with hand waving" minus "standing motion"] plus "walking motion."

ALGORITHM 1: Skeleton Augmentation

Input : $\mathcal{S} = \{S_1, S_2, \ldots, S_K\}$
Output: $\mathcal{S}' = \{S'_1, S'_2, \ldots, S'_H\}$

$\mathcal{S}' \leftarrow \{\}$
for $i \leftarrow 1$ *to* H **do**
  $S_i \leftarrow$ Randomly Select from $\mathcal{S}$;
  $S'_i \leftarrow$ `AddVariation`(S_i);
  Append S'_i to $\mathcal{S}'$;
end
return $\mathcal{S}'$

Function `AddVariation`(S):
  `// Copy the given skeletal structure`
  $S' \leftarrow S$;
  for *joint in* S'*.joints* **do**
    if *joint* $\in$ *{ Spine, Neck }* **then**
      randomly add or remove joints
    end
    if *joint* $\in$ *{ Hip, Shoulder }* **then**
      randomly add dummy joints
    end
    if *joint* $\in$ *{ End-Effectors }* **then**
      randomly set joint.offset as zero vector
    end
    randomly scale joint.offset
  end
  randomly scale S'.height;
  return S'

ALGORITHM 2: Motion Augmentation

Input : $\mathcal{M} = \{M_1, M_2, \ldots, M_N\}$ `// Motion DB`
  $\mathcal{S}' = \{S'_1, S'_2, \ldots, S'_H\}$ `// Augmented Skeleton DB`
Output: $\mathcal{M}' = \{M'_1, M_2, \ldots, M'_L\}$ `// Augmented Motion DB.`
`//` N`: size of input motion DB` $\mathcal{M}$
`//` B`: retarget batch size`
`//` R`: number of retargeting per each motion`
$\mathcal{M}' \leftarrow \{\}$
for $i \leftarrow 1$ *to* ceil (N/B) **do**
  Sample batch from $\mathcal{M}$
  for $j \leftarrow 1$ *to* R **do**
    Random Select from S'_i from $\mathcal{S}'$;
    Retarget batch into a target skeleton;
    Append retargeted batch into $\mathcal{M}'$;
  end
end
return $\mathcal{M}'$

A DATA COMPOSITION

Here, we outline the data employed for training and testing the base autoencoder as well as for animation tasks (Table 3). For training data, it presents the volume of data we originally collected which is augmented to generate training pairs (Section 4.3.1). For LAFAN1, PFNN, Mixamo, and SNU Actor DB, motion clips were randomly selected from their respective databases because they include many repetitive motions. We excluded motions involving object interactions, such as climbing stairs or lying on a bed. For lower-body reconstruction (Section 6.5.4), we fine-tuned the encoder using the same dataset we used for training base autoencoder model.

Table 3. Motion Data Composition

	Source	Size [min]	Subjects
AutoEncoder Train/Test			
Train (Section 4.3)	LAFAN1	41.72	1
	ACAAD	19.10	2
	TotalCapture	35.32	5
	PFNN	22.79	1
	SFU	11.73	7
Test (Section 6.4 and 6.6)	Mixamo Motion	28.04	9
	MotionBuilder	7.41	1
	SNU Actor DB	31.79	1
Motion Classification (Section 6.5.1)			
Train	CMU	45.28	37
Test	CMU	16.13	27
Similar Motion Search (Section 6.5.3)			
Search key	Mixamo Motion	8.99	1
Search DB	MotionBuilder	7.41	1
Finger Reconstruction (Section 6.5.4)			
Train	Mixamo Motion	7.09	1
Test	MotionBuilder	7.41	1
Interaction Character Control (Section 6.5.5)			
Motion Matching Database	LAFAN1	8.25	1
	PFNN	0.74	1
	SNU Actor DB	0.21	1

Table 4. Autoencoder architecture.

Name	Layers	In Dim	Out Dim	Heads
Encoder	GAT Conv + ReLU	32	16	16
	GAT Conv + ReLU	256	16	16
	GAT Conv + ReLU	256	16	16
	GAT Conv	256	32	1
	Max Pooling	32	32	-
Decoder	GAT Conv + ReLU	38	16	16
	GAT Conv + ReLU	262	16	16
	GAT Conv + ReLU	262	16	16
	GAT Conv	262	11	1

B DATA AUGMENTATION ALGORITHMS

We first augment the collected skeleton dataset by randomizing topology, body proportion, and height. Algorithm 1 depicts the pseudo-code of the process. Then, we generate training motion pairs by retargeting motion dataset to random skeletons (Algorithm 2).

C AUTOENCODER ARCHITECTURE

Table 4 shows each layer component and dimensions of our proposed autoencoder. We used multi-head graph attention layers (GATConv) with 16 heads and ReLU activation function. To aggregate joint features as a fixed-size latent vector, we used max pooling operator at the end of the encoder.